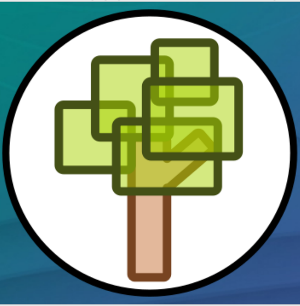
Karlsruhe Institute of Technology

# Faster Block Tree Construction

## 31st European Symposium on Algorithms (ESA 2023)

Dominik Köppl, *Florian Kurpicz*, and Daniel Meyer



# Motivation

- Git repositories
- DNA
- proteins
- user generated content W
- XML data
- book collections
- . . .

highly repetitive
&
huge input

# Motivation

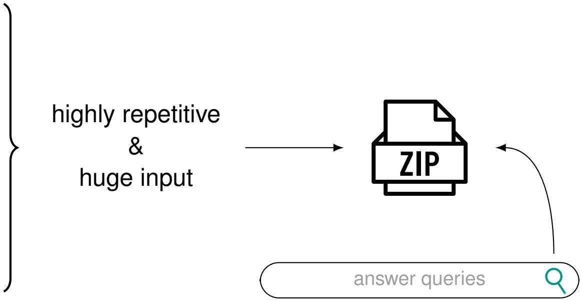
- Git repositories
- DNA
- proteins
- user generated content W
- XML data
- book collections
- . . .

highly repetitive
&
huge input

→ ZIP

# Motivation

- Git repositories
- DNA
- proteins
- user generated content
- XML data
- book collections
- . . .



highly repetitive

&

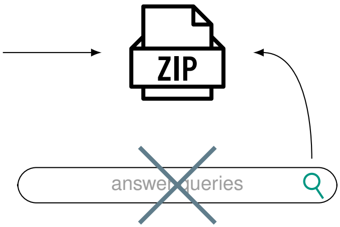
huge input

answer queries

# Motivation

- Git repositories
- DNA
- proteins
- user generated content W
- XML data
- book collections
- . . .

highly repetitive
&
huge input

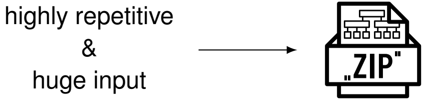


~~answer queries~~

# Compressed Self-Indices

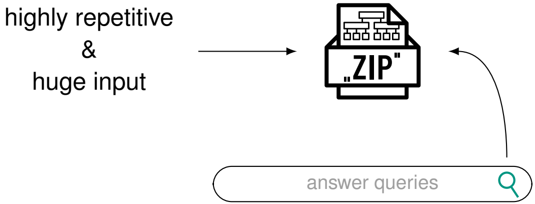
highly repetitive
&
huge input

# Compressed Self-Indices



highly repetitive
&
huge input

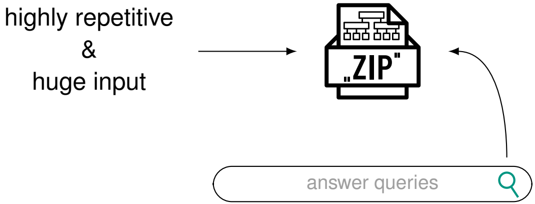
# Compressed Self-Indices



highly repetitive
&
huge input

answer queries

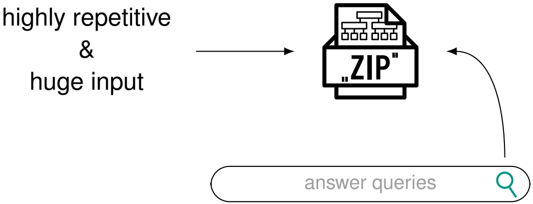
# Compressed Self-Indices



highly repetitive
&
huge input

- access
- rank
- select
- . . .
- pattern matching

# Compressed Self-Indices


highly repetitive

&

huge input

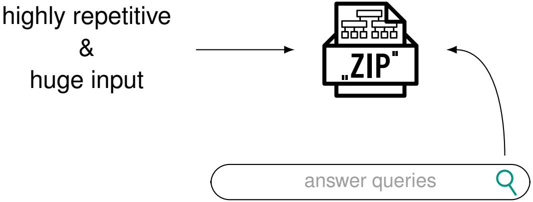
answer queries

- access
- rank
- select
- . . .
- pattern matching

**Wavelet Tree** (de-facto standard in practice)

- $T \in [1, \sigma]^n$
- access, rank, select: $O(\log \sigma)$ time
- $nH_0(T) + o(n)$ bits space

# Compressed Self-Indices



- access
- rank
- select
- . . .
- pattern matching

**Wavelet Tree** (de-facto standard in practice)

- $T \in [1, \sigma]^n$
- access, rank, select: $O(\log \sigma)$ time
- $nH_0(T) + o(n)$ bits space
- **blind for repetitions**

# Measures of Repetitiveness (Excerpt)

*b* < *z* < *g* < *r* < *e*

· # phrases in bidirectional macro scheme: *b*

| a | b | r | a | c | a | d | a | b | r | a |
|---|---|---|---|---|---|---|---|---|---|---|

(6,2)r(11,1)cad(1,4)

NP-hard

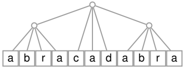
· # phrases in Lempel-Ziv parse: *z*

| a | b | r | a | c | a | d | a | b | r | a |
|---|---|---|---|---|---|---|---|---|---|---|

abr(1,1)c(1,1)d(1,4)

$O(n)$ time

· size of smallest context free grammar: *g*



NP-hard*

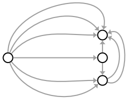
· # of runs in Burrows-Wheeler transform: *r*

| a | r | d | $ | r | c | a | a | a | a | b | b |
|---|---|---|---|---|---|---|---|---|---|---|---|

(1,a)(1,r)(1,d)(1,$)(1,r)(1,c)(4,a)(2,b)

$O(n)$ time

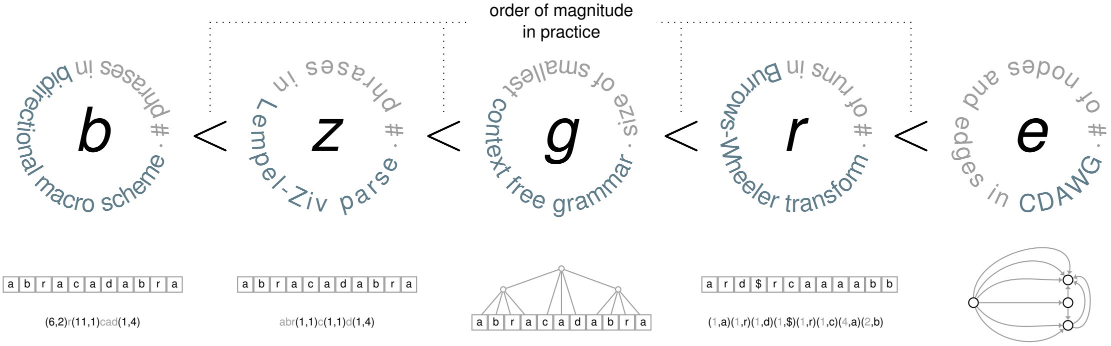
· # of nodes and edges in CDAWG: *e*



$O(n)$ time

* there are good heuristics

# Measures of Repetitiveness (Excerpt)


order of magnitude in practice
*b*
# phrases in bidirectional macro scheme
<
*z*
# phrases in Lempel-Ziv parse
<
*g*
size of smallest context free grammar
<
*r*
# of runs in Burrows-Wheeler transform
<
*e*
# of nodes and edges in CDAWG
a b r a c a d a b r a
(6,2)r(11,1)cad(1,4)
a b r a c a d a b r a
abr(1,1)c(1,1)d(1,4)
a b r a c a d a b r a
a r d $ r c a a a a b b
(1,a)(1,r)(1,d)(1,$)(1,r)(1,c)(4,a)(2,b)

NP-hard

$O(n)$ time

NP-hard*

$O(n)$ time

$O(n)$ time

* there are good heuristics

# Measures of Repetitiveness (Excerpt)

order of magnitude in practice

$b < z < g < r < e$

**$b$**: # · phrases in bidirectional macro scheme

| a | b | r | a | c | a | d | a | b | r | a |
|---|---|---|---|---|---|---|---|---|---|---|

(6,2)r(11,1)cad(1,4)

NP-hard

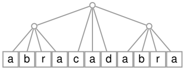
**$z$**: # · phrases in Lempel-Ziv parse

| a | b | r | a | c | a | d | a | b | r | a |
|---|---|---|---|---|---|---|---|---|---|---|

abr(1,1)c(1,1)d(1,4)

$O(n)$ time

**$g$**: size of smallest context free grammar ·



NP-hard*

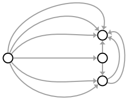
**$r$**: # · of runs in Burrows-Wheeler transform

| a | r | d | $ | r | c | a | a | a | a | b | b |
|---|---|---|---|---|---|---|---|---|---|---|---|

(1,a)(1,r)(1,d)(1,$)(1,r)(1,c)(4,a)(2,b)

$O(n)$ time

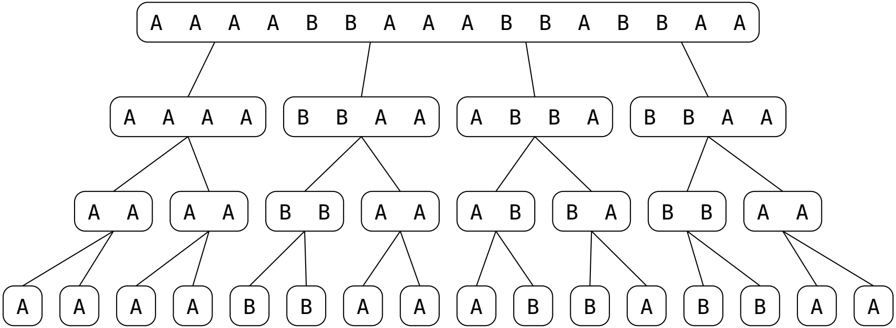
**$e$**: # · of nodes and edges in CDAWG



$O(n)$ time

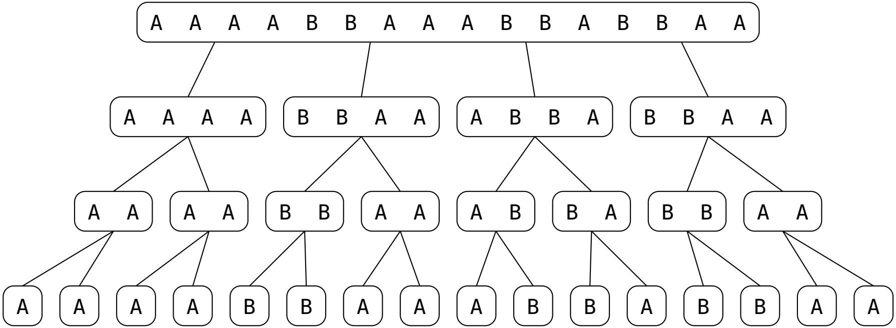
* there are good heuristics

# The Block Tree



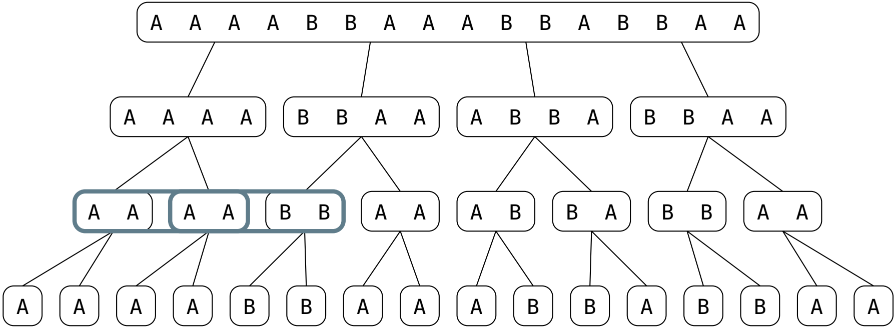
- root has degree $s = 4$
- other nodes have degree $\tau = 2$
- all levels (except the first) have $\leq 3z\tau$ blocks

# The Block Tree



- root has degree $s = 4$
- other nodes have degree $\tau = 2$
- all levels (except the first) have $\leq 3z\tau$ blocks
- keep $B_i$ if consecutive $B_{i-1} \cdot B_i$ or $B_i \cdot B_{i+1}$ are leftmost occurrences

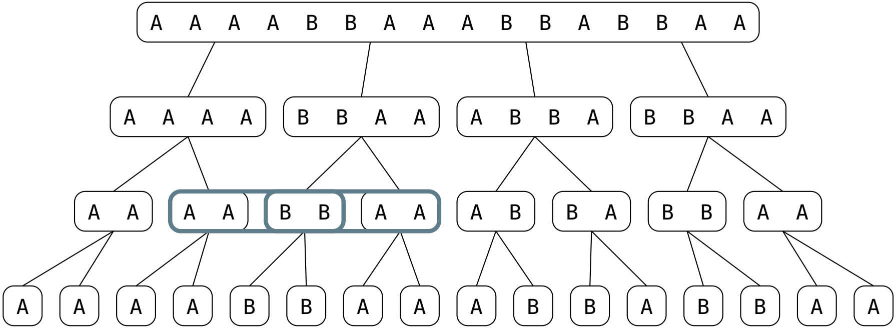
# The Block Tree



- root has degree $s = 4$
- other nodes have degree $\tau = 2$
- all levels (except the first) have $\leq 3z\tau$ blocks

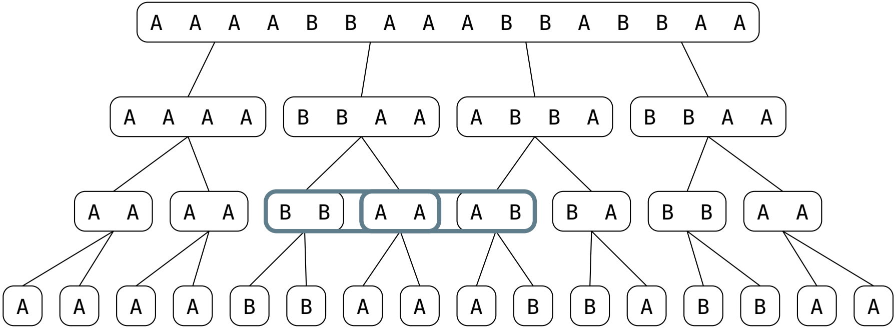
- keep $B_i$ if consecutive $B_{i-1} \cdot B_i$ or $B_i \cdot B_{i+1}$ are leftmost occurrences

# The Block Tree



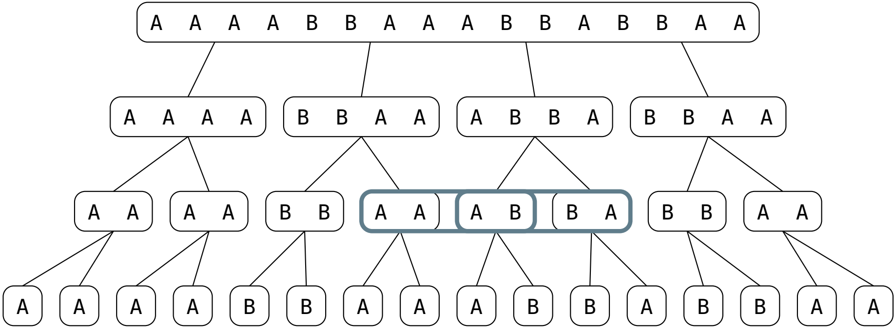
- root has degree $s = 4$
- other nodes have degree $\tau = 2$
- all levels (except the first) have $\leq 3z\tau$ blocks
- keep $B_i$ if consecutive $B_{i-1} \cdot B_i$ or $B_i \cdot B_{i+1}$ are leftmost occurrences

# The Block Tree



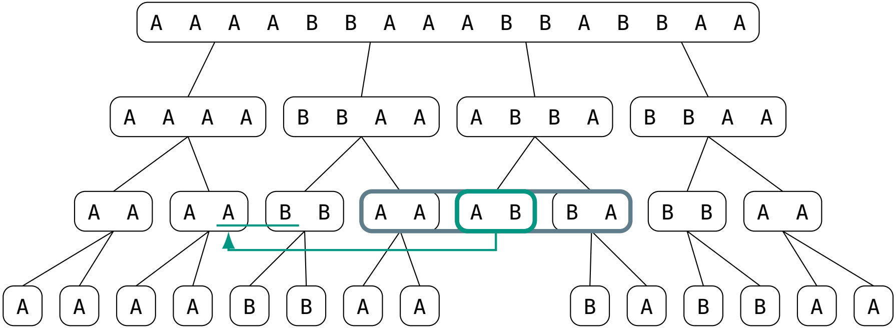
- root has degree $s = 4$
- other nodes have degree $\tau = 2$
- all levels (except the first) have $\leq 3z\tau$ blocks
- keep $B_i$ if consecutive $B_{i-1} \cdot B_i$ or $B_i \cdot B_{i+1}$ are leftmost occurrences

# The Block Tree



- root has degree $s = 4$
- other nodes have degree $\tau = 2$
- all levels (except the first) have $\leq 3z\tau$ blocks
- keep $B_i$ if consecutive $B_{i-1} \cdot B_i$ or $B_i \cdot B_{i+1}$ are leftmost occurrences

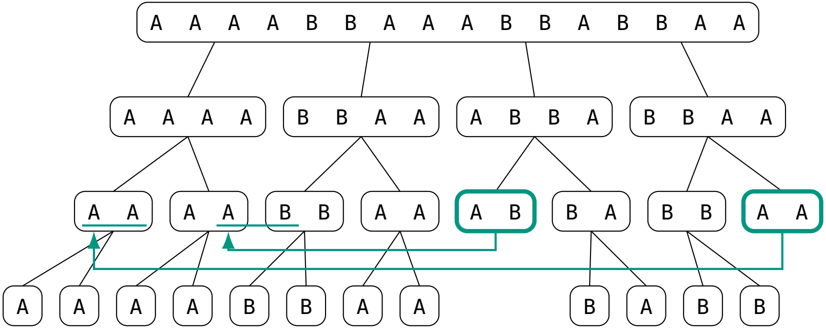
# The Block Tree



- root has degree $s = 4$
- other nodes have degree $\tau = 2$
- all levels (except the first) have $\leq 3z\tau$ blocks

- keep $B_i$ if consecutive $B_{i-1} \cdot B_i$ or $B_i \cdot B_{i+1}$ are leftmost occurrences

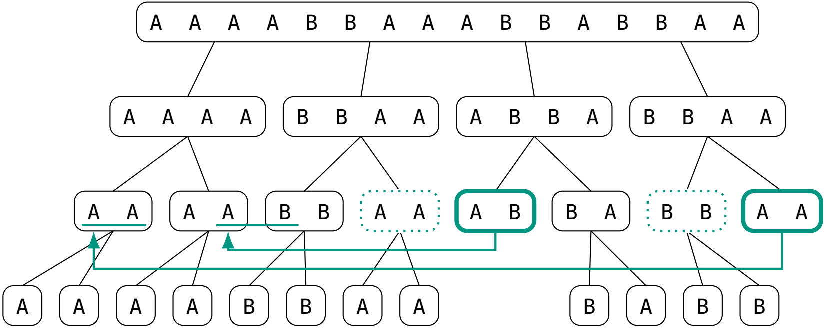
# The Block Tree



- root has degree $s = 4$
- other nodes have degree $\tau = 2$
- all levels (except the first) have $\leq 3z\tau$ blocks

- keep $B_i$ if consecutive $B_{i-1} \cdot B_i$ or $B_i \cdot B_{i+1}$ are leftmost occurrences

# The Block Tree



- root has degree $s = 4$
- other nodes have degree $\tau = 2$
- all levels (except the first) have $\leq 3z\tau$ blocks
- keep $B_i$ if consecutive $B_{i-1} \cdot B_i$ or $B_i \cdot B_{i+1}$ are leftmost occurrences
- remove more blocks with pruning

# State-of-Block-Tree-Construction

| Method | Reference | Working Space | Time | Implementation |
|---|---|---|---|---|
| Aho-Corasic | [Bel+21] | $O(n)$ | $O(n(1+\log_\tau(z\tau/s)))$ | no |
| Fingerprints | [Bel+21] | $O(s+z\tau\log_\tau(\frac{n\log\sigma}{s\log n}))$ | $O(n(1+\log_\tau(z\tau/s)))$ expected | yes (slow) |
| LPF Array | [here] | $O(n)$ | $O(n(1+\log_\tau(z\tau/s)))$ | yes (fast) |

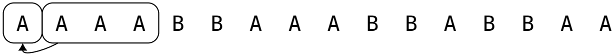
# Lempel-Ziv Parse

A A A A B B A A A B B A B B A A

# Lempel-Ziv Parse

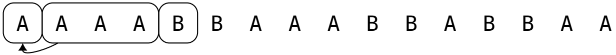
$\boxed{\text{A}}$ A A A B B A A A B B A B B A A

# Lempel-Ziv Parse



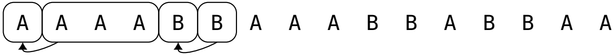
A A A A B B A A A B B A B B A A

# Lempel-Ziv Parse

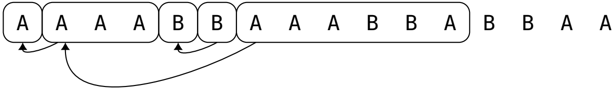


A A A A B B A A A B B A B B A A

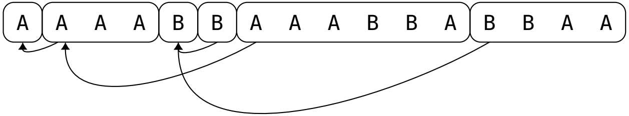
# Lempel-Ziv Parse



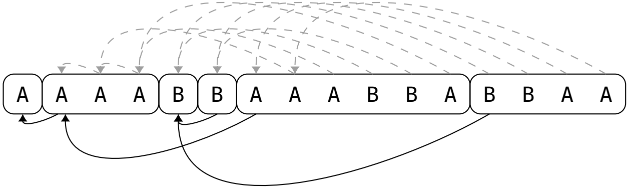
# Lempel-Ziv Parse



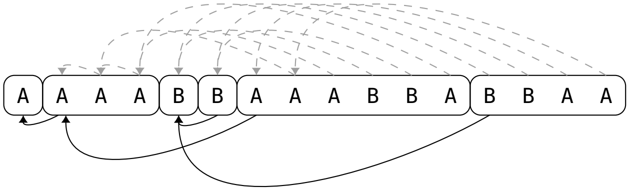
# Lempel-Ziv Parse



# Lempel-Ziv Parse

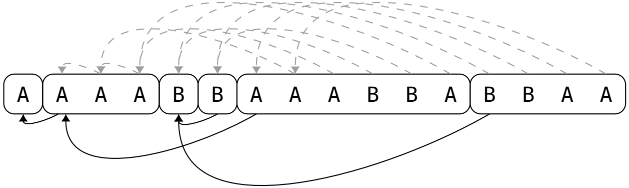


# Lempel-Ziv Parse



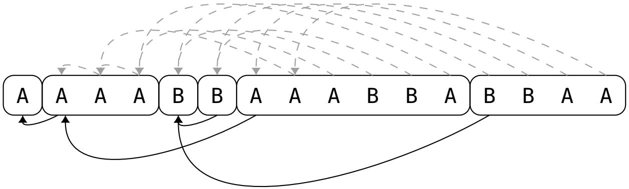
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | A | A | A | A | B | B | A | A | A | B | B | A | B | B | A | A |
| LPF | 0 | 3 | 2 | 1 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 | 2 | 1 |

# Lempel-Ziv Parse



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | A | A | A | A | B | B | A | A | A | B | B | A | B | B | A | A |
| LPF | 0 | 3 | 2 | 1 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 | 2 | 1 |
| PrevOcc | -1 | 1 | 2 | 3 | -1 | 5 | 2 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 7 | 8 |

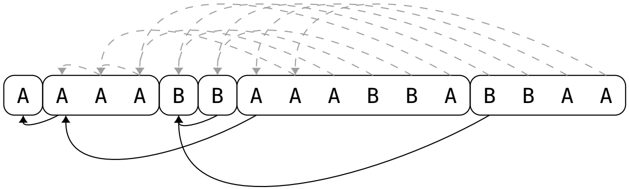
# Our Algorithm (Marking of Nodes)



| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | A | A | A | B | B | A | A | A | B | B | A | B | B | A | A |
| LPF | 0 | 3 | 2 | 1 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 | 2 | 1 |
| PrevOcc | -1 | 1 | 2 | 3 | -1 | 5 | 2 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 7 | 8 |

A A　A A　B B　A A　A B　B A　B B　A A

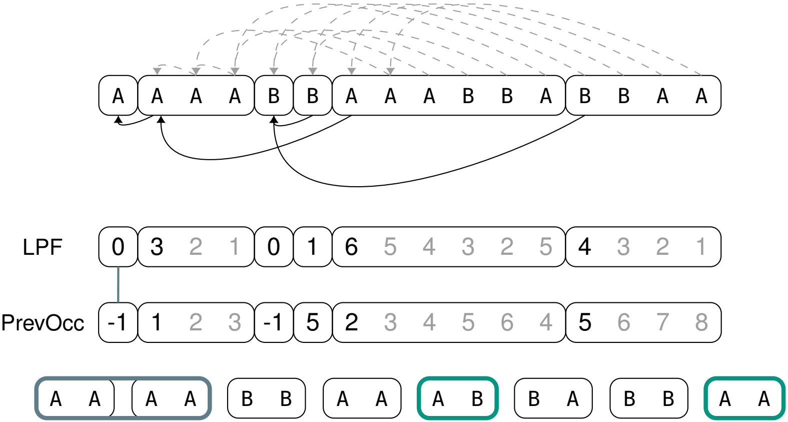
# Our Algorithm (Marking of Nodes)



| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LPF | 0 | 3 | 2 | 1 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 | 2 | 1 |
| PrevOcc | -1 | 1 | 2 | 3 | -1 | 5 | 2 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 7 | 8 |

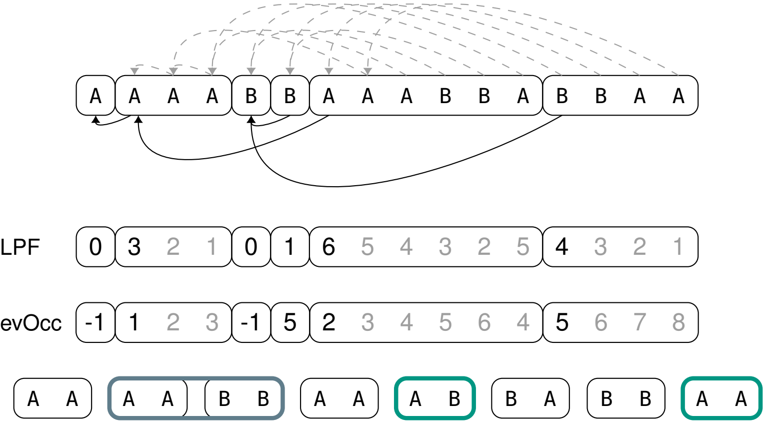
A A A A | B B | A A | A B | B A | B B | A A

# Our Algorithm (Marking of Nodes)

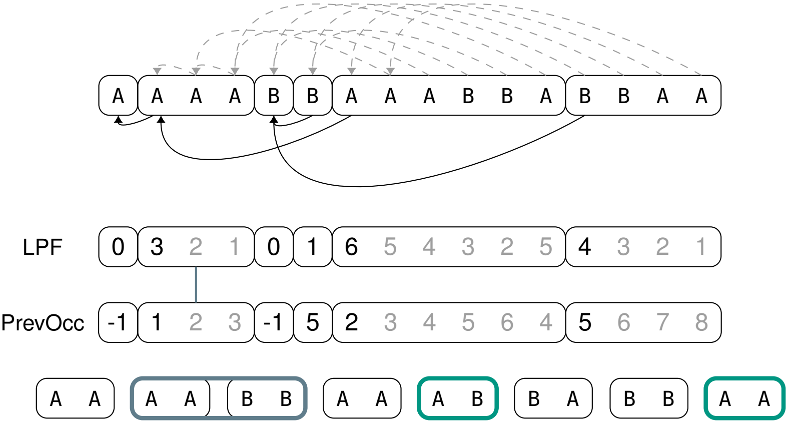


|  | A | A | A | A | B | B | A | A | A | B | B | A | B | B | A | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LPF | 0 | 3 | 2 | 1 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 | 2 | 1 |
| PrevOcc | -1 | 1 | 2 | 3 | -1 | 5 | 2 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 7 | 8 |

# Our Algorithm (Marking of Nodes)



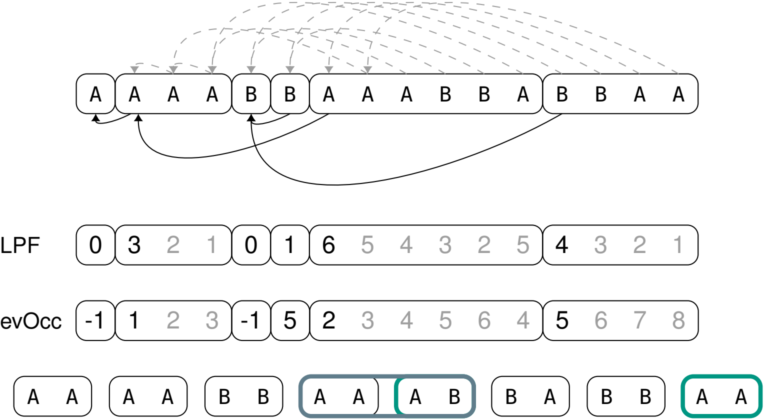
# Our Algorithm (Marking of Nodes)



| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | A | A | A | B | B | A | A | A | B | B | A | B | B | A | A |
| LPF | 0 | 3 | 2 | 1 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 | 2 | 1 |
| PrevOcc | -1 | 1 | 2 | 3 | -1 | 5 | 2 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 7 | 8 |

A A   A A B B   A A   A B   B A   B B   A A

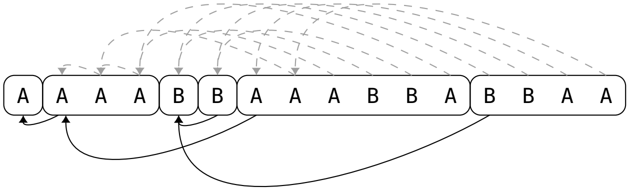
# Our Algorithm (Marking of Nodes)



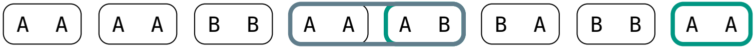
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | A | A | A | A | B | B | A | A | A | B | B | A | B | B | A | A |
| LPF | 0 | 3 | 2 | 1 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 | 2 | 1 |
| PrevOcc | -1 | 1 | 2 | 3 | -1 | 5 | 2 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 7 | 8 |

A A | A A | B B | A A | A B | B A | B B | A A

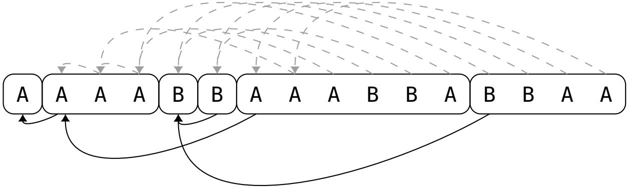
# Our Algorithm (Marking of Nodes)



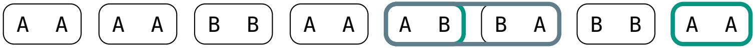
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LPF | 0 | 3 | 2 | 1 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 | 2 | 1 |
| PrevOcc | -1 | 1 | 2 | 3 | -1 | 5 | 2 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 7 | 8 |



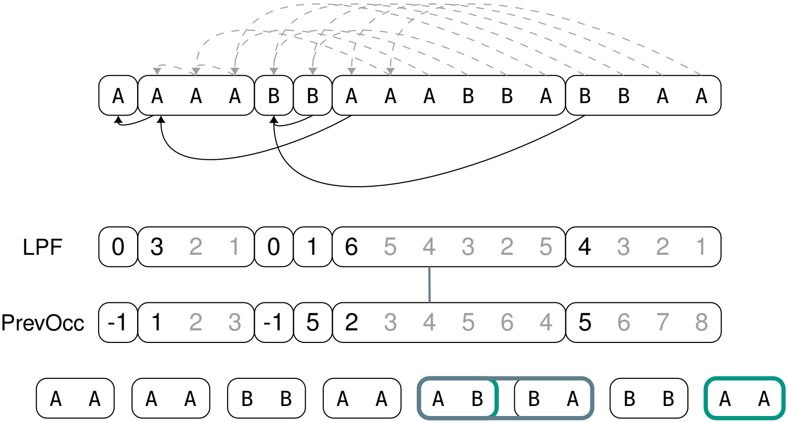
# Our Algorithm (Marking of Nodes)



| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LPF | 0 | 3 | 2 | 1 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 | 2 | 1 |
| PrevOcc | -1 | 1 | 2 | 3 | -1 | 5 | 2 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 7 | 8 |



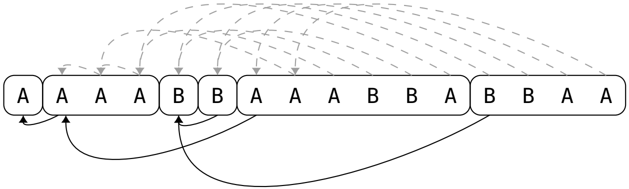
# Our Algorithm (Marking of Nodes)



| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | A | A | A | B | B | A | A | A | B | B | A | B | B | A | A |
| LPF | 0 | 3 | 2 | 1 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 | 2 | 1 |
| PrevOcc | -1 | 1 | 2 | 3 | -1 | 5 | 2 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 7 | 8 |

A A | A A | B B | A A | A B | B A | B B | A A

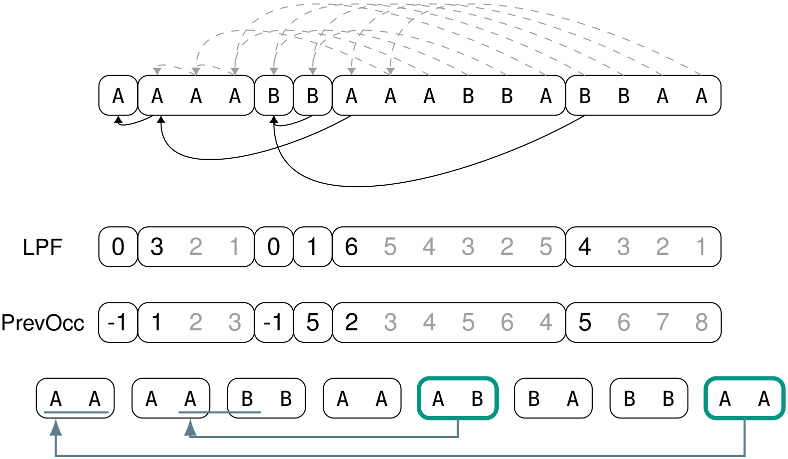
# Our Algorithm (Marking of Nodes)



| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | A | A | A | B | B | A | A | A | B | B | A | B | B | A | A |
| LPF | 0 | 3 | 2 | 1 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 | 2 | 1 |
| PrevOcc | -1 | 1 | 2 | 3 | -1 | 5 | 2 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 7 | 8 |

A A　A A　B B　A A　A B　B A　B B　A A

# Our Algorithm (Marking of Nodes)



|  | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | A | A | A | A | B | B | A | A | A | B | B | A | B | B | A | A |
| LPF | 0 | 3 | 2 | 1 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 5 | 4 | 3 | 2 | 1 |
| PrevOcc | -1 | 1 | 2 | 3 | -1 | 5 | 2 | 3 | 4 | 5 | 6 | 4 | 5 | 6 | 7 | 8 |

# Experimental Evaluation

- highly tuned implementation
- tree consists only of bit and compact vectors
- tuning parameter
  - degree root $s = \{1, z\}$ (only we have $s = z$)
  - degree other nodes $\tau = \{2, 4, 8, 16\}$
  - number characters in leaves $b = \{2, 4, 8, 16\}$

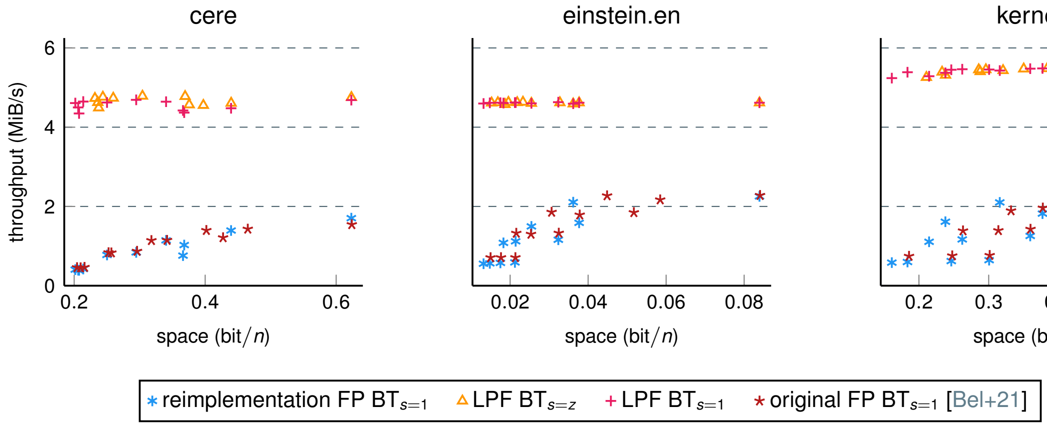
# Experimental Evaluation

- highly tuned implementation
- tree consists only of bit and compact vectors
- tuning parameter
  - degree root $s = \{1, z\}$ (only we have $s = z$)
  - degree other nodes $\tau = \{2, 4, 8, 16\}$
  - number characters in leaves $b = \{2, 4, 8, 16\}$

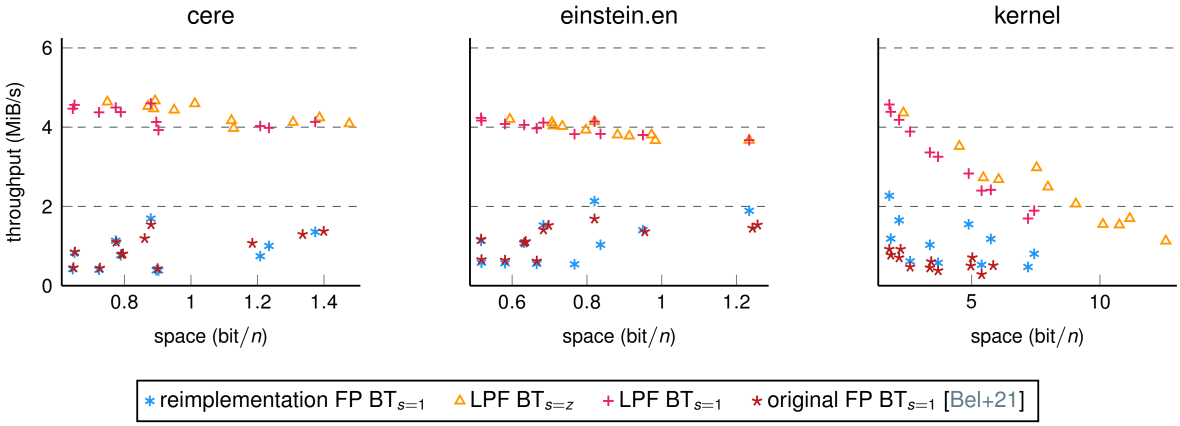
- original FP BT [Bel+21]
- our reimplementation of the original FP BT
- our LPF BT construction with $s = 1$ and $s = z$
- dynamic programming variants
- parallelization
- no comparison with wavelet trees (faster)

- repetitive instances from P&C corpus
- non-repetitive instances from P&C corpus

# Highly Repetitive Inputs (Access Only)



# Highly Repetitive Inputs (with Rank and Select Support)



# Conclusion and Future Work

- fastest block tree construction algorithm
- first parallel block tree construction
- works in practice for non-repetitive inputs

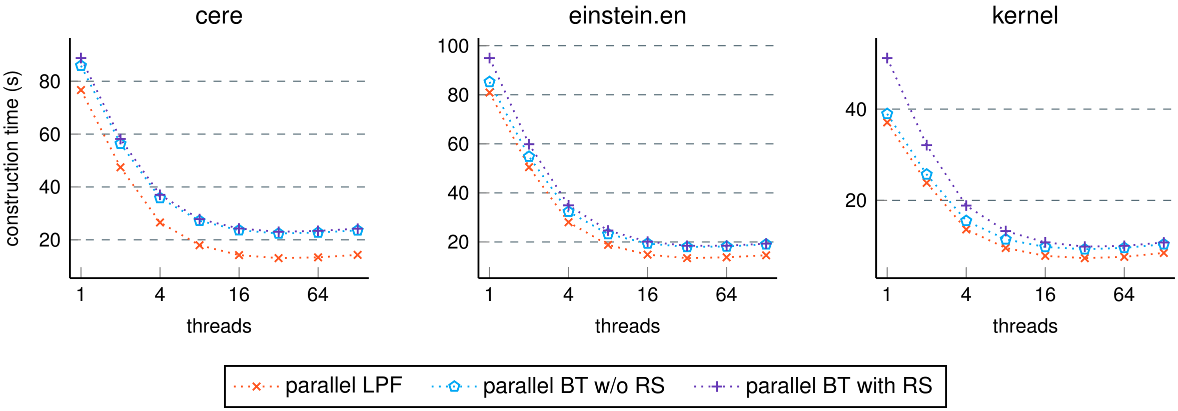
- better scaling parallel construction
- better marking rules (less pruning)

/pasta-toolbox/block_tree

erc
European Research Council
Established by the European Commission

This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 882500).

# Parallel Construction (Strong Scaling)



# Bibliography

[Bel+21] Djamal Belazzougui, Manuel Cáceres, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Gonzalo Navarro, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. “Block Trees”. In: *J. Comput. Syst. Sci.* 117 (2021), pages 1–22. DOI: 10.1016/j.jcss.2020.11.002.